08/09/2023

* PASS BY VALUE →

- When passed by value, a copy is created in the function that is to be called of the variable present in the main function.
- Both have different addresses irrespective of their same variable names or may be different names.
- When we made changes in the variable that is copied in the called function, these changes will not reflect upon the variable present in main function.

Example -1

```
#include <iostream>
using namespace std;

passByValue (int a){          → Copy created [50]
    a--;        → 49
    a/=7;       → 7
    cout<<a<<endl;  → Print "7"
}

int main(){
    int a=9;
    a++;        → 10
    a*=5;       → 50
    passByValue(a);
    cout<<a<<endl;  → Print "50"
}
```

Output = 7
50

Example-2

```
#include <iostream>
using namespace std;

passByValue (int m){
    m--;
    m*=10;
    cout<< m <<endl;
}

int main( ){
    int marks =90;
     marks ++;
    passBy Value (marks);
    cout << marks <<endl;
}
```

Output = 900
91

NOTE- The variable marks in main function is copied to variable m in passByValue function.

Example 3-

```
#include <iostream>
using namespace std;

passByValue (int Jaadu){
    Jaadu--;
    cout<< Jaadu <<endl;
}

int main( ){
    int sundari =100;
     Sudari--;
     sundari-=5;
     passByValue (sundari);
     cout<< sundari<< endl;
}
```

Output = 93
94

Note- The variable sundari in main function is copied to variable Jaadu in passBy Value function.

NOTE → In pass by Reference, ~~same~~ different variable name is given to same memory location.

* **PASS BY REFERENCE →**

- When passed by reference, no copy is created but we are working on the actual memory location in the called function.
- Both have same address i.e. the variable in main function and called function. whose variable names may be same or different.
- When we made changes in the variable present in the called function, these changes will reflect upon the variable present in the main function.
- ampersand symbol (&) is used for passing by reference.

Example-1

```
#include<iostream>
using namespace std;

void passByRefer(int &b){
    b--;
    cout<<b+5<<endl;
}
int main(){
    int a=9;
    a++;
    passByRefer(a);
    cout<<a<<endl;
}
```

Output= 14
9

Example → 2

```
#include<iostream>
using namespace std;

void passByRefer (int &m){
    m--;
    m*=10;
    cout<<m<<endl;
}
int main(){
    int marks=90;
    marks++;
    passByRefer(marks);
    cout<<marks<<endl;
}
```

Output = 900
900

Example 3 –

```
#include<iostream>
using namespace std;

void passByRefer (int &Jaadu){
    Jaadu--;
cout<< Jaadu+10<< endl;
}
int main (){
    int sundari=100;
    sundari--;
    sundari -=5;
passByRefer (sundari);
    cout<<sundari<<endl;
}
```

Output = 103
93

Note – The scope of sundari is only in main function and scope of variable Jaadu is only in passByRefer function. We can't access them in another function.

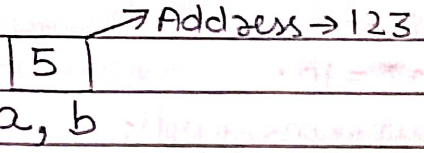
NOTE - `int &n = 6;` → error

We can't assign any constant value to the reference variable.

* **REFERENCE VARIABLES →**

Same memory locations but different variable names.

Let,



We can access this memory location by both variables a and b. Variable b doesnot have ~~their~~ its existence in memory as own.

Example →

```
#include<iostream>
using namespace std;

int main(){
  int a=5;
  int &b=a;
  int &c=b;
  a++;
  b+=2;
  c*=10;

  cout<<a<<" "<<b<<" "<<c<<" ";
}
```

Output = 80 80 80

Here we can see that we can access variable a using b and c too using reference variables concept.

NOTE → Whenever we pass an array across function, the address of that array is sent, not the actual array copy.
If it were the actual array, then the copy of array will have formed which will waste too much space in the memory.
So, by default, when we pass an array across function, it is passed by Reference.

Example - **Pass By Reference in Array →**

```
#include <iostream>
using namespace std;

void passByRefer(int arr[], int n){
    arr[2] = 30;
}
int main(){
    int arr[] = {10,20,25,40};
    int size = 4;
    passByRefer(arr, size);

    for(int i=0; i<size; i++){
        cout << arr[i] << " ";
    }
}
```

Output = 10 20 30 40

Note → int arr[] → Passed by reference by default.
int n → Passed by value.

* Find unique element →

I/P = 2, 10, 11, 13, 10, 2, 15, 13, 15

Here, each element is occuring twice except one i.e. 11.

It can be solved using XOR operator.

```
#include<iostream>
using namespace std;

int getUnique (int arr[], int n){
    int ans= 0;
    for(int i=0; i<n; i++){
        ans = ans ^arr[i];
    }
    return ans;
}
int main( ){

    int arr[ ]={2,10,11,10,2,15,13,15,15};
    int n = 9;
    int finalAns = getUnique (arr,n);
    cout<<"Final answer: "<< finalAns <<endl;
}
```

Output = 11

* Print all pairs for the elements present in array →

```
#include <iostream>
using namespace std;

int main( ){
  int n= 3;
  int arr[ ]= {10, 20, 30};

  for (int i=0 ; i<n ; i++){
    for (int j=0 ; j<n ; j++){
      cout<< arr[i]<<","<< arr[j] <<endl;
    }
   cout<< endl;
  }
}
```

Output =

```
10, 10
10, 20
10, 30

20, 10
20, 20
20, 30

30, 10
30, 20
30, 30
```

* Sorting zeroes and ones in array →

```
#include<iostream>
using namespace std;

void sortZeroOne (int arr[] ,int n){

  // for Counting zeroes and ones
    int zeroCount = 0;
    int oneCount = 0;
    for (int i=0; i<n ; i++){
        if (arr[i]==0){
            zeroCount++;
        }
        else if (arr[i]==1){
            oneCount++;
        }
    }

  // for sorting 0's and 1's respectively

    int index=0;
    while (zeroCount--){
      arr[index]=0;
      index++;
    }
    while(oneCount--){
        arr[index]=1;
        index++;
    }
}
```

Note → zeroCount-- and oneCount-- will run until their count becomes 0.

__/__/__

```
int main(){
    int arr[] = {0,1,1,0,0,1,1,1,1};
    int n = 9;

    SortZeroOne(arr,n);

    // for printing the sorted array

    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }
}
```

output = 0 0 0 1 1 1 1 1 1

__/__/__

* Shift array by 1 →

let, I/P = 10, 20, 30, 40, 50, 60
O/P = 60, 10, 20, 30, 40, 50

```
#include <iostream>
using namespace std;

void shiftArray (int arr[], int n){

  // Store arr[n-1]
    int temp = arr[n-1];

  // Shift → arr[i] = arr[i-1]
   for (int i=n-1; i>0; i--){
       arr[i] = arr[i-1];
      }

  // Copy temp. into 0th index
     arr[0] = temp;
  }
int main(){
    int arr[] = {10,20,30,40,50,60};
    int n= 6;

    shiftArray (arr,n);

    // Printing the output array
    for (int i=0; i<n; i++){
        cout<< arr[i]<<" ";
      }
```

Output = 60, 10, 20, 30, 40, 50

__/__/____

* Shift array by 1 →

Let, I/P = 10, 20, 30, 40, 50, 60
O/P = 20, 30, 40, 50, 60, 10

```
#include <iostream>
using namespace std;

void shiftArray (int arr[], int n){
    int temp = arr[0];

    // Shift → arr[i] = arr[i+1]
    for (int i=0; i<n-1; i++){
        arr[i] = arr[i+1];
    }

    // copy temp into (n-1)th index
        arr[n-1] = temp;
}
int main(){
    int arr[] = {10, 20, 30, 40, 50, 60};
    int n = 6;
    shiftArray (arr, n);

    // Printing the output array
    for (int i=0; i<n; i++){
        cout<< arr[i] <<" ";
    }
```

Output = 20, 30, 40, 50, 60, 10